View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Calhoun, Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

Theses and Dissertations | Thesis Collection

1993-09

NPSNET: scripting of three-dimensional interactive systems for use in the Janus combat simulation

Smith, Richard Samuel.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/40001

http://www.nps.edu/library

Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A274 862

DTIC
ELECTE
JAN 24 1994
S E D

THESIS

NPSNET: SCRIPTING OF THREE-DIMENSIONAL INTERACTIVE SYSTEMS FOR USE IN THE JANUS COMBAT SIMULATION

by

Richard Samuel Smith

September 1993

Thesis Advisor: Dr. David R. Pratt
Co-Advisor: Dr. Michael J. Zyda

Approved for public release; distribution is unlimited.

5196 94-01944

94 1 21 126

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1993	Master's Thesis

4. TITLE AND SUBTITLE
NPSNET: SCRIPTING OF THREE-DIMENSIONAL INTERACTIVE SYSTEMS FOR USE IN THE JANUS COMBAT SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Smith, Richard Samuel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problems addressed in this thesis are that the combat simulator Janus does not generate combat scenarios in a three-dimensional interface, and that Janus scenarios are incompatible with the three-dimensional NPSNET. Janus' two-dimensionality led in some cases to less realistic path and position selections, and more realism was desired for evaluations of combat systems. The approach to solving the problems was to generate a script file for NPSNET from a Janus combat scenario. Then the scripted scenario was run on NPSNET where interactions between combat systems could be observed in a three-dimensional virtual battlefield. Entities could be maneuvered in NPSNET to create more realistic paths. The maneuvers were written to a script which was then merged with the original Janus scenario. The result of this work is six programs which assemble a Janus scenario into an NPSNET script file; and two programs which write the results of the NPSNET maneuvers into the original Janus scenario. With these programs users can develop or evaluate Janus scenarios from the more realistic perspective of a soldier on the battlefield rather than from an artificial perspective above a two-dimensional battlefield. Also combat systems can be evaluated in a more realistic environment. These results provided greater realism for an existing combat simulator.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Janus, NPSNET, combat scenario, virtual battlefield, scripting	52
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Approved for public release; distribution is unlimited

NPSNET: SCRIPTING OF THREE-DIMENSIONAL INTERACTIVE SYSTEMS FOR USE IN THE JANUS COMBAT SIMULATION

by

Richard S. Smith
Captain, United States Army
Bachelor of Fine Arts, Arizona State University, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

[redacted]
Author: Richard Samuel Smith

[redacted]
Approved By: Dr. David R. Pratt, Thesis Advisor

[redacted]
Dr. Michael J. Zyda, Thesis Co-Advisor

[redacted]
Dr. Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The problems addressed in this thesis are that the combat simulator Janus does not generate combat scenarios in a three-dimensional interface, and that Janus scenarios are incompatible with the three-dimensional NPSNET. Janus' two-dimensionality led in some cases to less realistic path and position selections, and more realism was desired for evaluations of combat systems.

The approach to solving the problems was to generate a script file for NPSNET from a Janus combat scenario. Then the scripted scenario was run on NPSNET where interactions between combat systems could be observed in a three-dimensional virtual battlefield. Entities could be maneuvered in NPSNET to create more realistic paths. The maneuvers were written to a script which was then merged with the original Janus scenario.

The result of this work is six programs which assemble a Janus scenario into an NPSNET script file; and two programs which write the results of the NPSNET maneuvers into the original Janus scenario. With these programs users can develop or evaluate Janus scenarios from the more realistic perspective of a soldier on the battlefield rather than from an artificial perspective above a two-dimensional battlefield. Also combat systems can be evaluated in a more realistic environment. These results provided greater realism for an existing combat simulator.

DTIC QUALITY INSPECTED 5

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Dave Young helped me on many occasions to understand the use of the graphics tools. Kevin Walsh helped me through my first programming class. John Locke always had time to discuss the Janus system he admired so much, and to help me trouble shoot my code. Mark Declue was also helpful in debugging my code and solving some of the vehicle motion problems. I also thank Dr. Pratt for his encouragement throughout my time at NPS, from the first class I took from him until the end of my thesis work. His chants of "Work! Work! Work!" were, of course, inspirational to all of his thesis students.

I. INTRODUCTION

A. STATEMENT OF PROBLEM

The problem was that the Janus combat simulator was only two-dimensional. This thesis was a proof of concept design of a three-dimensional interface for Janus. With the three-dimensional interface, the user could do path planning and line of sight estimates from the perspective of a soldier seeing the battlefield rather than from the perspective of a point above a two-dimensional battlefield. This three-dimensional capability was intended to enhance scenario generation, modification and evaluation by providing a more realistic view of the battlefield.

Even though present and future military spending cutbacks will reduce the number of large scale training exercises the military can afford, the need for realistic training will not diminish. Realistic training is essential for our military units to maintain a high state of combat readiness. Realistic training more often than not means three-dimensional and interactive training.

Combat simulation systems have already proven to be valuable and economical alternatives to live exercises. Computer simulated training exercises do not use real fuel or live ammunition to move and shoot in cyberspace. Training in cyberspace does not pollute the environment. Further, the participants in the exercises spend more time training than deploying to and from field locations. However, many of these useful combat simulation systems were designed to display the battlefield in two-dimensions which, at the time of their development, was state of the art. Upgrading these proven training tools to allow the users to view the battlefield in three dimensions for combat scenario development or evaluation is a cost effective way of making good combat simulation systems more realistic.

The Computer Science Department at the Naval Postgraduate School in Monterey, California has developed a low-cost battlespace simulation system, known as NPSNET,

which displays objects in a three-dimensional interactive space [Zyda92]. NPSNET is designed to work on off-the-shelf Silicon Graphics IRIS workstations.

The United States Army Training and Doctrine Command uses a ground combat simulation system called Janus for training, research, and testing of combat systems. In 1992 at the Naval Postgraduate School, CPT Pat Warren USA and CPT Jon Walter USA integrated Janus with NPSNET so that an existing Janus terrain database could be displayed in a three-dimensional virtual reality world. [Walt92]

The goal of this research is to develop a tool that translates Janus initialization data files into a script suitable for the three-dimensional, interactive environment of NPSNET. And then to translate back into the Janus data files the scenario developed in NPSNET. This goal is represented in Figure 1.

B. SUMMARY OF CHAPTERS

Chapter II provides an overview of NPSNET. Chapter III provides an overview of Janus. Chapter IV discusses the translation of Janus binary data to an NPSNET input script file. Chapter V covers the translation of an NPSNET output script file into the Janus binary data files. Chapter VI is a summary of conclusions and further work. Appendix A contains the user guide for the Janus/NPSNET scripting system.

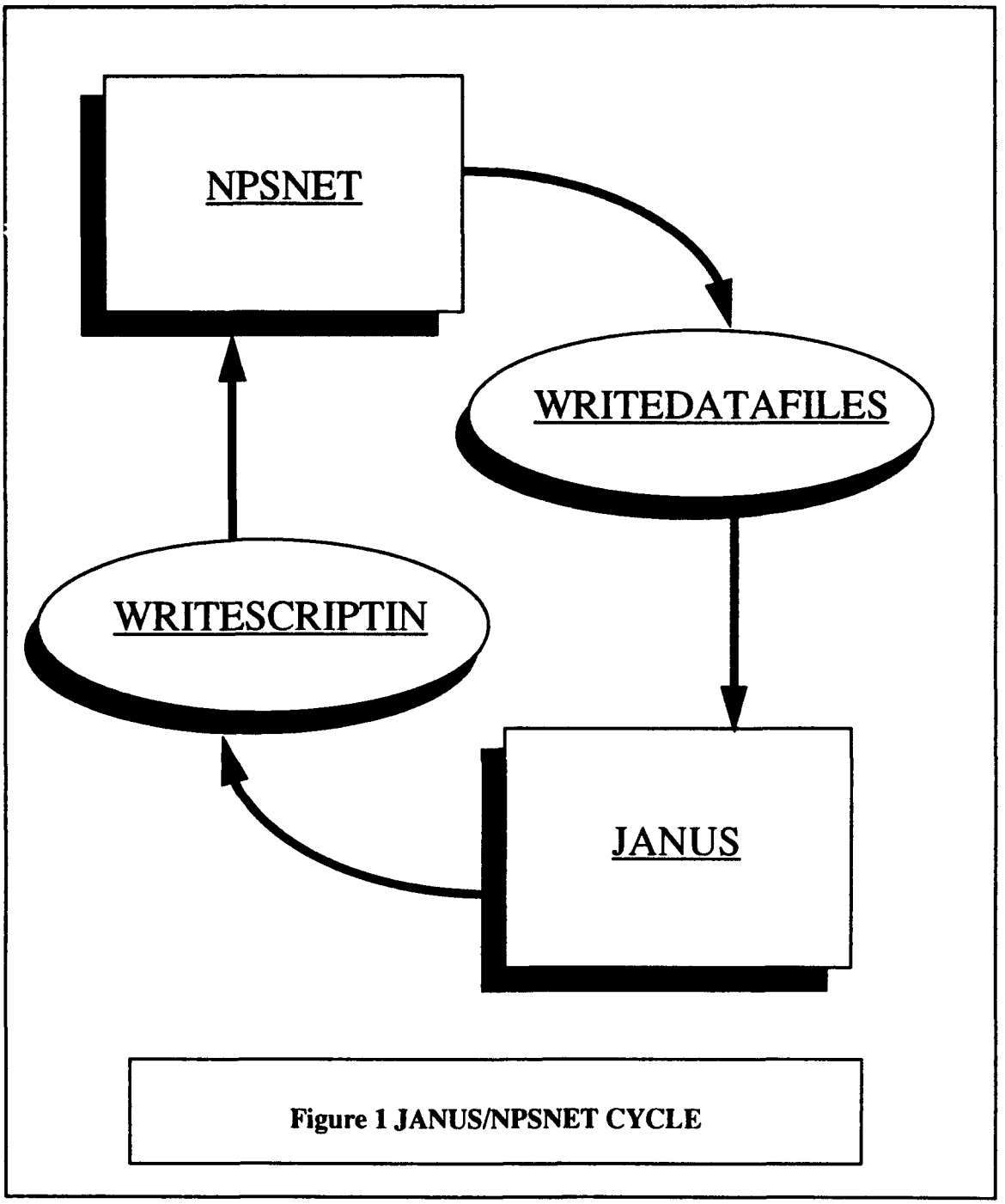

Figure 1 JANUS/NPSNET CYCLE

II. OVERVIEW OF NPSNET

NPSNET is a real-time, three-dimensional visual simulation system, currently under development at the Computer Science Department of the Naval Postgraduate School (NPS) in Monterey, California [Zyda92]. NPSNET is designed to be a low-cost simulation system run on Silicon Graphics, Inc. IRIS workstations with the IRIX (UNIX based) operating system [SGI 91].

NPSNET is a totally interactive battle simulation system in which the user can select any one of 500 different active vehicles and control it with several devices, including a six degree of freedom SpaceBall, a keyboard, a mouse and a button/dialbox. Other vehicles in the simulation are controlled by users on other workstations, expert systems, or by NPSNET itself.

Different modes of combat modeling were available in NPSNET such as control, script, and network. First, the models can be maneuvered in the three-dimensional battlefield by a player on one machine interface. The second mode of combat modeling uses an Ethernet network for communication and interaction between local workstations where NPSNET broadcasts locally designed packets. For large scale interaction at many different levels, a translator has been implemented which provides the capability to transmit packets compatible with the Simulation Networking (SIMNET) protocol. Current work includes the design of an expanded translator which is compatible with the Distributed Interactive Simulation (DIS) protocol. [Zyda93]

Scripting was another modeling mode of NPSNET where scenarios of combat operations could be viewed in three dimensions. This was the mode used in this thesis. When in the scripting mode, NPSNET used script files to store combat scenario information. Script files could also be generated by NPSNET wherein entity actions were stored. Actions were represented in the script files by lines of data where each line, or series of lines, represented an event. The script line format consisted of fifteen fields which contained all the data NPSNET required to display the entity at a specific time, place and

orientation. So a turn by a vehicle for example could be represented by one or more script lines.

Vehicles and objects in NPSNET are modeled using the locally developed NPS Object File Format (NPSOFF) [Zyda93]. NPSOFF is an ASCII formatted language which incorporates many Graphics Library (GL) [SGI91] function calls into a single object file. The overall format of NPSOFF closely resembles a series of standard GL calls. By representing objects in this way, NPSOFF provides a simple method of encapsulating objects which are easily transportable between programs and can be referenced in an abstract manner. An ASCII format also makes the file readable and modifiable with any text editor. Ongoing work on NPSNET models includes implementation of the Graphics Description Language (NPSGDL) which is a C++ based system to further encapsulate models and their properties [Wils92].

III. JANUS OVERVIEW

Janus is a United States Army interactive, computer based, war-gaming simulation of brigade and lower level unit combat operations. The original Janus simulation began in the late 1970's at the Lawrence Livermore National Laboratory (LLNL) to model nuclear effects. It gained a considerable reputation for innovative use of graphical user interfaces. The U. S. Army Training and Doctrine Analysis Command, White Sands Missile Range, New Mexico (TRAC-WSMR), acquired this prototype from LLNL as a result of the Janus Acquisition and Development Project directed by the U. S. Army Training and Doctrine Command in 1980. In 1983, TRAC-WSMR adopted Janus and further developed it as a high resolution simulation to support analysis for Army combat developments. [Walt 92]

The original version developed at LLNL is known as Janus(L), and the model developed by TRAC-WSMR is known as Janus(T). Both of these models gained in popularity and were employed by a large number of users, which led to the proliferation of different versions of the models. The Janus(Army) Program began in 1989 to solve the standardization problem and to field a single version - Janus(A). Today Janus(A) is developed, maintained and distributed by TRAC-WSMR and is fielded throughout the world as a tool for both trainers and analysts in research, testing and combat development. [Walt 92]

Janus(A) is a "two-sided", interactive, closed, stochastic, ground combat simulation. Janus is "two-sided" since it simulates two opposing forces - the blue force and the red force - simultaneously directed and controlled by players on separate monitors. It is termed 'closed' since players do not know the complete disposition of opposing forces - each monitor displays only the vehicles on its side and the opposing force vehicles which can be observed from its vehicles. The model is 'interactive' because each player monitors, directs, reacts to, and redirects all key actions of the simulated units under his control. Once a scenario is started, certain events in the game, such as direct fires and artillery impacts, are stochastically modeled, which means the events act according to the laws of probability,

and thus are slightly different for every scenario run. The principal modeling focus in Janus(A) is on military systems that participate in maneuver and artillery operations on land, thus the term 'ground combat simulation'. [Walt 92]

Initially, Janus was run on any Digital Equipment Corporation VAX family of computers using the standard VMS operating system. In 1991, the U. S. Army directed that Janus(A) be fielded on an "open system". In April, 1992 Jim Guyton of the Rand Corporation delivered a working prototype of Janus (A) for UNIX. This new version, called Janus(X) is identical to Janus(A) except that FORTRAN calls to Tektronix screens are replaced with "C" programming language calls to the X-Windows environment. [Walt 92]

Janus(A) is composed entirely of Army-developed algorithms and data to model combat processes. The multitude of programs which belong to Janus(A) consist of approximately 200,000 lines of code written entirely in VAX-11 FORTRAN, a structured Digital Equipment Corporation extension of ANSI standard FORTRAN-77 . In addition to these combat simulation programs, Janus(A) also has eleven utility programs to facilitate the creation, running, and after-action analysis of a specific scenario. [Walt 92]

Previous integration of NPSNET with Janus post-processor files was accomplished by CPT Pat Warren and CPT Jon Walter in 1992 at the Naval Postgraduate School. Their integration of Janus with NPSNET concentrated on the output of Janus: the post processor files. Their work is represented in Figure 2.

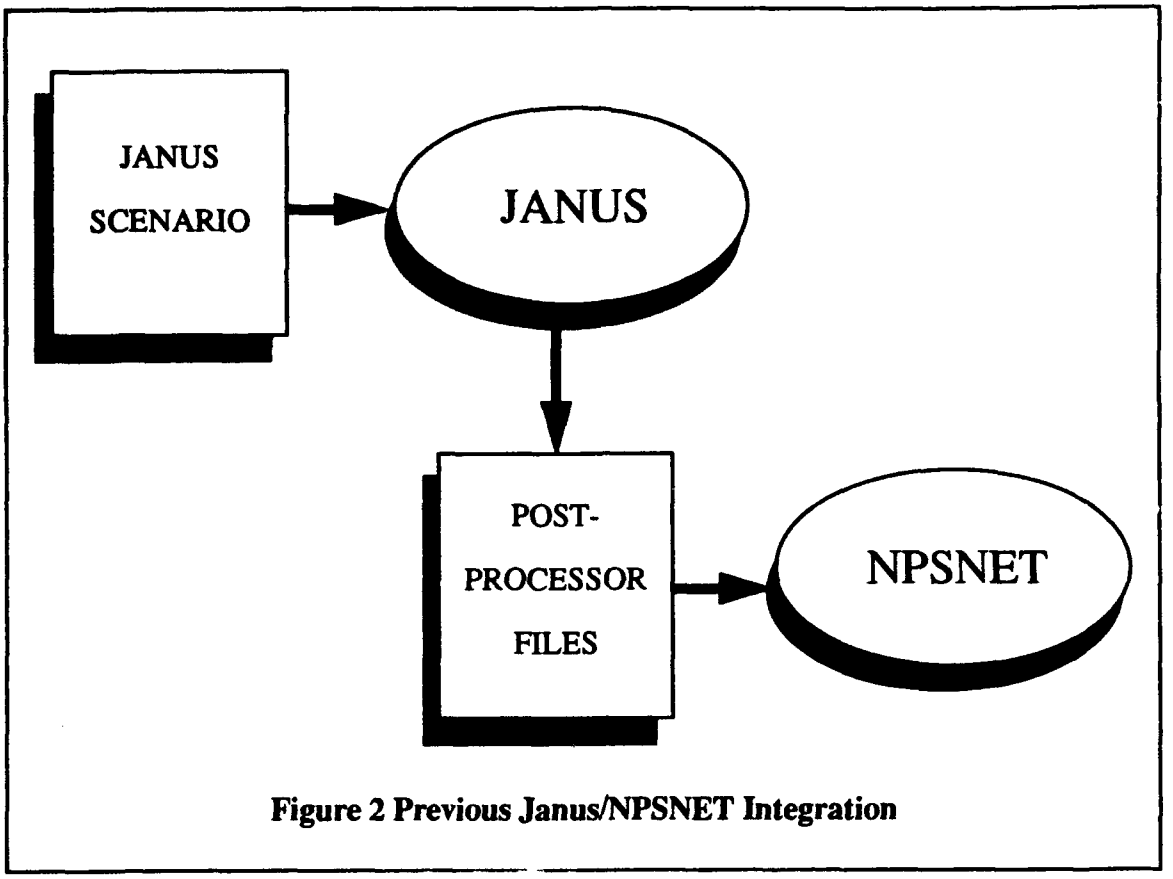

Figure 2 Previous Janus/NPSNET Integration

IV. BUILDING A NPSNET SCRIPT FILE FROM A JANUS SCENARIO

A. THE DATA FILES

The four binary data files SYSTMXXX.DAT, DPLOYXXX.DAT, FORCXXX.DAT, and JSCRNXXX.DAT described a Janus scenario (where the XXX represent the digits of a scenario number). Of the four binary files which made up a Janus, scenario only three were needed to build an NPSNET script file: SYSTMXXX.DAT, DPLOYXXX.DAT, and FORCXXX.DAT. No post-processing information was used.

Each of the files was composed of many arrays of data in binary format. In order to understand how each binary data file was organized, the FORTRAN subroutines that read each type of data file had to be translated. The read subroutines told the order of data in each file and how many arrays there were. For example, the subroutine DPLOYREAD told that XUNIT was the first of thirty-one arrays in the DPLOYXXX.DAT data file, Table 1. FORCREAD listed fourteen, and SYSTMREAD listed two hundred thirty-six arrays, Tables 2 and 3. A few of the Janus parameters tell a lot about the system. The parameter

Table 1: ARRAYS IN DPLOYXXX.DAT JANUS DEPLOY FILE

Name of Array	Array Element Type	Size of Array	Offset of Arrays Used (bytes)[a]
XUNIT	real numbers	600x2	4
YUNIT	real numbers	600x2	4804
MOUNTED	short integers	600x2	
KHOLFIR	bytes	600x2	
MOPP	bytes	600x2	
KDEFL	bytes	600x2	
MAXPREP	short integers	2	
NBRPREP	short integers	2	
PREPX	real numbers	600x2	

Table 1: ARRAYS IN DPLOYXXX.DAT JANUS DEPLOY FILE

Name of Array	Array Element Type	Size of Array	Offset of Arrays Used (bytes)[a]
PREPY	real numbers	600x2	
XNODE	real numbers	50x600x2	25236
YNODE	real numbers	50x600x2	265236
IOBJPTR	integers	600x2	
KTIMENODE	short integers	51x600x2	506436
KFLAGNODE	bytes	51x600x2	
DVIEW	real numbers	600x2	693644
DLEFT	real numbers	600x2	
DRIGT	real numbers	600x2	
DFORM	real numbers	600x2	
KOBTYPE	byte	1	
KOBSIDE	byte	1	
XOBST	real numbers	2x200	
YOBST	real numbers	2x200	
[illegible]IELDS	byte	1	
MINEFLDS	integers	16x50	
KSPRINT	bytes	600x2	
KMQNEXT	integer	1	
KBTRYSMSN	integers	200	
KBTRYTMSN	integers	200	
KBTRYTIME	bytes	200	
KQMSN	integers	10x12x200	

a. Only the values of interest to this thesis are entered.

NUMUNITS has a value of 600, which is the number of systems each side can have. NUMSIDES has the value of two - there are two sides in Janus. NUMNODES has the value

of 50, which is the maximum number of way points which may be entered for each system. Many of the arrays are set up NUMUNITS * NUMSIDES, which means each data element in the array corresponds to a particular system.

Table 2: ARRAYS IN FORCXXX.DAT JANUS FORCE FILE

Name of Array	Array Element Type	Size of Array	Offset of Arrays Used (bytes)
KNUMUNITS	short integers	2	0
KSYSTYP	short integers	600x2	4
KCSDTYP	short integers	600x2	
KNUMELE	bytes	600x2	
KTASKFOR	bytes	600x2	
KMCLRFUN	bytes	600x2	
KMCLRLOD	bytes	600x2	
KMINLOD	short integers	600x2	
KCLASSF	bytes	20	
KPHPKFILE	byte	1	
KWTHTYP	byte	1	
KSIMSYS	short integers	50x2	
KSIMWPN	short integers	50x2	
STUDYNAM	character	16	

Table 3: ARRAYS IN SYSTMXXX.DAT JANUS SYSTEM FILE

Name of Array[a]	Array Element Type	Size of Array	Offset of Arrays Used (bytes)
WETHRNAM	character	1	
SYSTNAME	character	50x2	16
WEAPNAME	character	50x2	

a. The only the first three of the 236 arrays are not listed for brevity.

But these FORTRAN read programs did not tell how big each array was. To get the size of each array numerous forces files such as GLBUNITS.FOR and GLOBDIR.FOR, which listed the parameters of each array, had to be researched. So, for example, to find the size of the array XUNIT in the DPLOYXXX.DAT data file, look in the GLBUNITS.FOR file which lists the global variables that are the parameters of XUNIT: NUMUNITS and NUMSIDES. The values of the parameters might sometimes be found in GLBPARAM.FOR or sometimes the parameter might be listed in the forces file that defined the array.

Finally, the size of the data type had to be determined. By convention, variable names in FORTRAN beginning with I,J,K,L,M,N,O were integers, and variable names beginning with other letters were not integers. Still there were further distinctions to be made for integers such as whether the integer was long or short. The point was to determine the length in bytes of each data unit in the arrays. Long integers and real numbers used four bytes, short integers used two bytes, and char a single byte unit of data. The single byte, and short integers were most often defined at the beginning of the forces file in which they were used.

B. READING THE JANUS FILES

When NPSNET was in the scripting mode, all the information required by NPSNET to place and orient a system at a designated point in time was given in a script line. Each line corresponded to a system at a starting point or a system at a node along the path the system was to follow. An NPSNET script file was made up of script lines and each line had fifteen fields of information. Figure 3 shows the fields of an NPSNET script line.

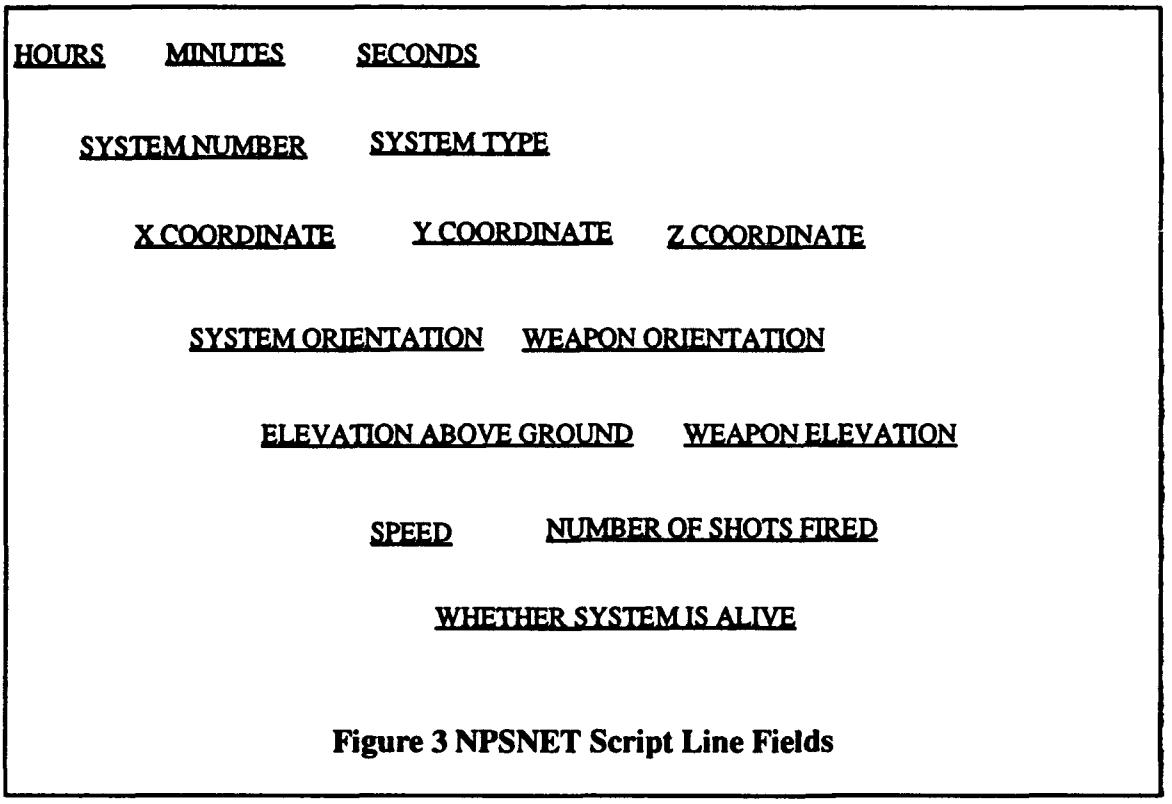

Figure 3 NPSNET Script Line Fields

Three processes readsundeploy, readsunforce, and readsunsystem took data from the Janus scenario files and stored the information in intermediate files. The process writescriptin assembled the data into an NPSNET script file.

Once it was determined which data arrays were to be used, the beginning of each array had to be determined so data could be read from the array, and so that data could be written back into the array. For example, in the data file DPLOYXXX.DAT, the data in the arrays

XUNIT, YUNIT, XNODE, YNODE, and DVIEW were needed. Table 1 above showed the DPLOYXXX.DAT arrays in order with their corresponding parameter types and sizes.

With the above information, it was a simple though tedious task to determine how many bytes the pointer at the beginning of the data file must traverse to get to any data element in any array. To assist in determining and confirming where data elements were in the vast binary initialization files, the system command od, octal dump, was very useful. This command dumps the contents of a binary file in one of several formats specified on the command line. The most useful format was -f which interpreted long words as floating point. This od command was also particularly useful since, in a few cases, the documentation for the array types was nebulous or contradictory, which made calculations of how many bytes to traverse to a particular array element deep in a binary file very tricky. With the od command, it was sometimes possible to determine by observation of the binary data where one type of data ended and another began which confirmed calculations of where particular types of data began or ended.

1. Readsundeploy

This process took as an argument the name of a binary data file of the form DPLOYXXX.DAT where the XXX represented the number of a particular Janus scenario. Janus systems have their initial node coordinates, subsequent node coordinates, node times, and initial orientations in DPLOYXXX.DAT. Figure 4 shows how readsundeploy contributed to the assembly of a NPSNET script.

a. XUNIT

XUNIT was the name of the Janus array for the x coordinates of the starting points of the 1200 systems. This data was found at the beginning of the binary data file, at the fourth byte, and was written to XUNIT_DAT.

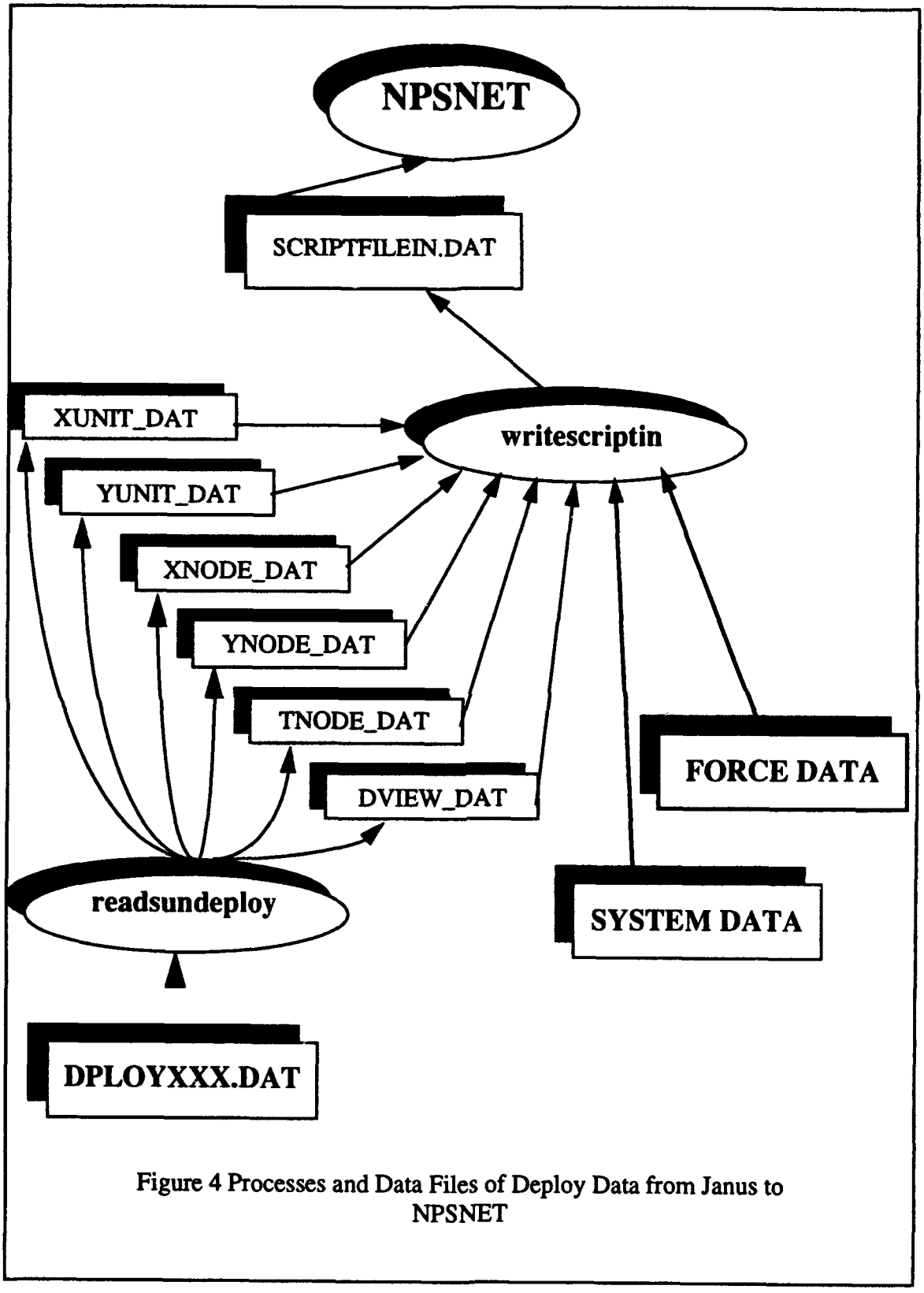

Figure 4 Processes and Data Files of Deploy Data from Janus to NPSNET

b. YUNIT

YUNIT was the name of the Janus array for the y coordinates of the starting points of the 1200 systems. This data was the second array of data elements in DPLOYXXX.DAT. It was written to YUNIT_DAT file.

c. XNODE

The XNODE array contained the x coordinates of the 50 possible path node points of each system - so the array had 60000 elements. The XNODE array was the eleventh array of data in DPLOYXXX.DAT and its values were written to the XNODE_DAT file.

d. YNODE

The YNODE array contained the y coordinates of the 50 possible path node points of each system - so the array had 60000 elements. The Ynode data was the twelfth array in the binary data file and was written to YNODE_DAT.

e. KTIMENODE

The KTIMENODE array contained the time in seconds from the start of the scenario for each of the path node points of each system. The array had 61200 entries - an entry for every start time and node time of all 1200 possible systems. For example, if the 20th byte of the array was 77 that would mean that the twentieth node of the first vehicle would be reached at one minute and 17 seconds. KTIMENODE was the fourteenth array of data in DPLOYXXX.DATand was written to the TNODE_DAT file. Note that this set of data was unlike Xnode and Ynode data since it included the times of the starting and subsequent positions: starting x and y coordinates were stored in separate data structures from x and y subsequent path node coordinates.

f. DVIEW

The DVIEW array contained the initial view directions in radians of the 1200 systems. DVIEW was the sixteenth array of data in DPLOYXXX.DATand was written to the DVIEW_DAT file.

2. Readsunforce

This process took as an argument the name of a binary data file of the form FORCXXX.DAT. The only array of data used in this binary data file was the KSYSTYP array which was written into the SYSTYP_DAT file. Figure 5 shows how readsunforce contributed to the NPSNET script file.

3. Readsunsystem

This process took as an argument the name of a binary data file of the form SYSTMXXX.DAT. The only data array used in this file was the system names array SYSTMNAMES. The array was the second in the file and each element was a string name. There were fifty possible system names for each side. Figure 6 shows how readsunsystem contributed to the NPSNET script file.

When considering the KSYSTYPE array in FORCXXX.DAT, we saw that the array size was NUMUNITS * NUMSIDES which meant the array consisted of 1200 elements. And each element corresponded to a particular system. The value in the array was a number which corresponded to the name of the system. In SYSTMXXX.DAT, the second array was called SYSTMNAMES, which listed the string name which corresponded to each number. Each side was allowed fifty different system names, so the array SYSTMNAMES had one hundred entries. The first fifty corresponded to one side, the second fifty corresponded to the other side. For instance in scenario 716, the number 17 on the blue side corresponded to a M1A1 tank, and on the red side the number 17 corresponded to a BRDM-M.

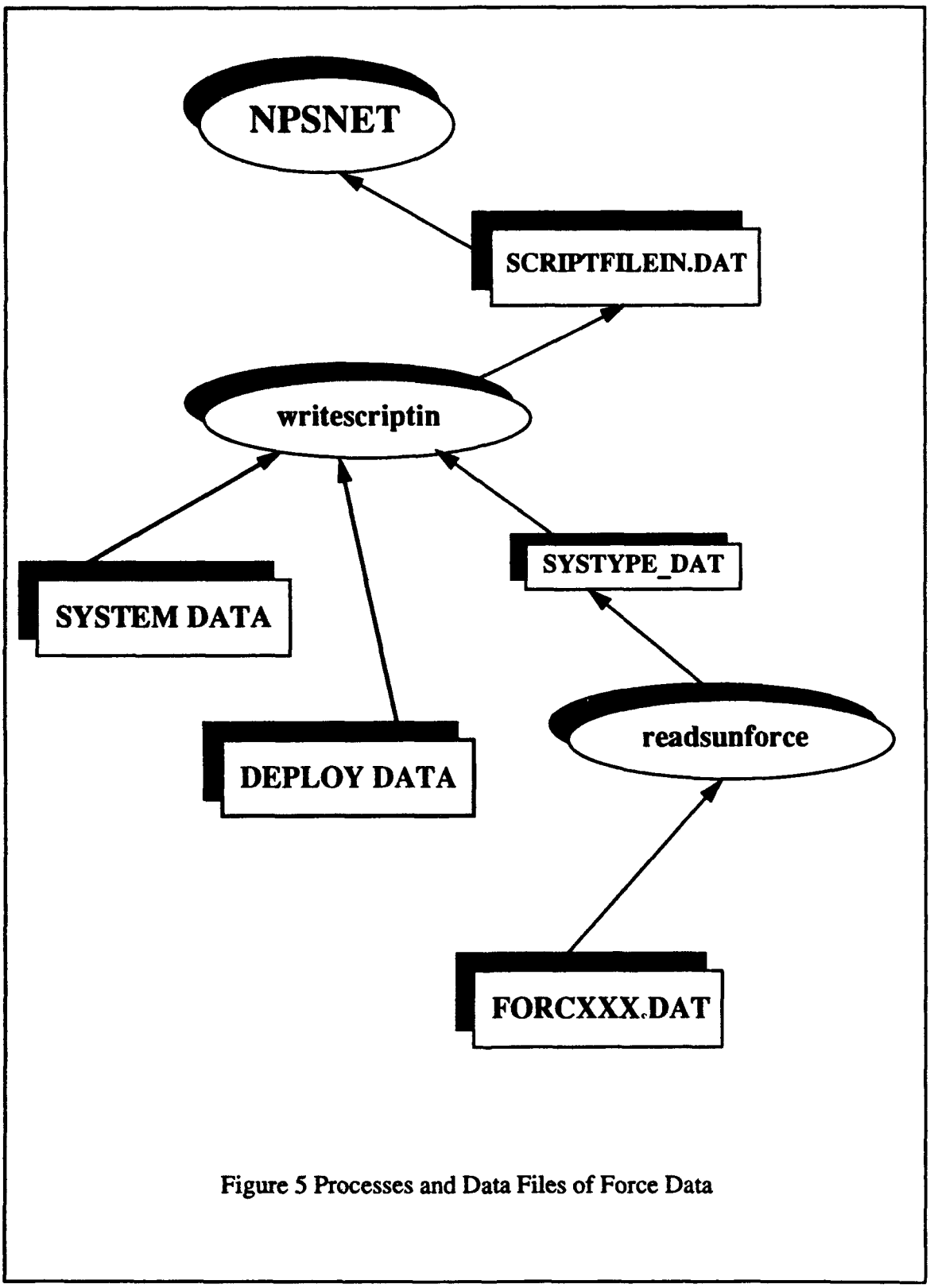

Figure 5 Processes and Data Files of Force Data

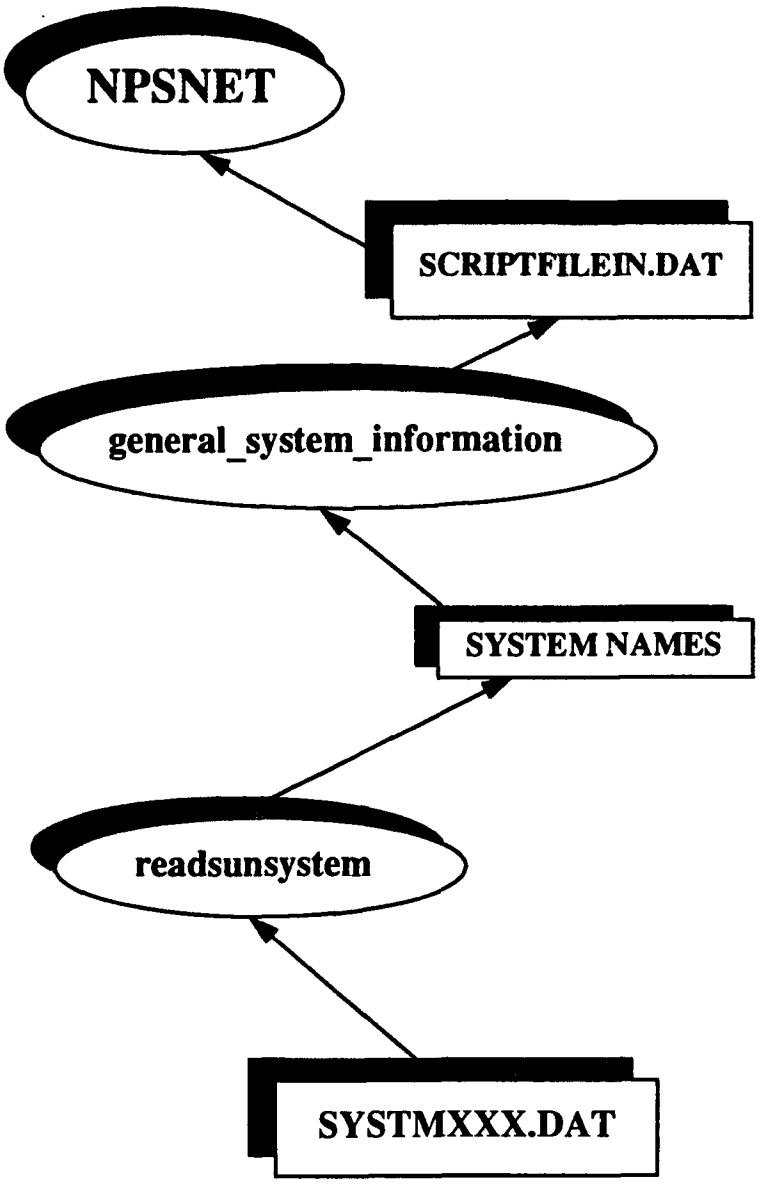

Figure 6 Processes and Data Files of System Data

C. ASSEMBLING THE DATA INTO AN NPSNET SCRIPT

The separate ASCII data files were "read" by function calls inside two loops. The outer loop corresponded to each system and the inner loop corresponded to paths of each system. So the inner loop could iterate fifty times for each iteration of the outer loop if all 1200 systems were used and each system had a path with fifty nodes. The data was processed and finally written to the file called scriptfilein.dat which was the script file used by NPSNET. The following paragraphs describe the processes that converted Janus data into NPSNET script files.

1. Maxandmin

This process found the maximum and minimum x and y coordinates from the XUNIT_DAT, XNODE_DAT, YUNIT_DAT, YNODE_DAT files. These values were used in the conversion of the coordinates from the UTM (universal transverse mercator) system used by Janus to world coordinates that NPSNET understood. The conversion factor was derived by finding the range from maximum to minimum in both x and y, taking the larger range, and dividing it into the size of the NPSNET terrain data base. The conversion factor depended on the NPSNET terrain data base used. For example, the conversion factor for the Fulda Gap scenario was 240: The range of both x and y values was 50, and the size of the Fulda Gap terrain data base in NPSNET was 12,000. This maxandmin process was done after readsundeploy and before writescriptin.

2. Main process of writescriptin

Each iteration of the outer loop of writescriptin read data from five intermediate files: xunit_dat, yunit_dat, tnode_dat, systype_dat, and dview_dat. Each iteration provided the data for the initial node of the twelve hundred systems. The Janus scenario data base had 0.0 for the values of the initial x and y coordinates of unused systems. This data was printed to the script lines which corresponded to the unused vehicles. The values of the x and y coordinates on the scanned lines were tested, and if they were greater than or equal to the minimum x and y values determined by the process "maxandmin", then the scanned

lines contained useful information. Scanned script lines which did not contain useful information were not processed. All the information about a particular system could have been contained in only one script line or there could have been fifty subsequent script lines each containing subsequent node information.

a. Processing initial node data

The following were the steps taken to process initial node data:

(1) Janus time was total time in seconds from the start of the scenario. NPSNET used hours, minutes, and seconds. So the total time given in seconds was converted and stored in buffers named hours, minutes and seconds. The values in the hours, minutes and seconds buffers were written in that order to the script file npsscriptin.

(2) The system number was the same as the number of the counter of the outer loop, and it was stored in a buffer call sys_num. The sys_num buffer content was written to npsscriptin next.

(3) The system type given by Janus was an integer which corresponded to a set of system characteristics. The function general_system_information took the system type number as its argument and assigned the system name, maximum speed, and corresponding NPSNET number to the buffers systypebuf, max_system_speed, and npsnet_sys_type_num. The npsnet_sys_type_num buffer content was written next to npsscriptin.

(4) The x coordinate given by Janus was converted to a coordinate that could be usefully displayed on NPSNET. This was done with a SCALE factor which was defined at the beginning of writescriptin.c. The factor was derived by dividing the difference between the maximum and minimum values of the scenario into the width of the terrain data as displayed on NPSNET. In the case of the Fulda Gap scenario, the difference was 50. The width of the Fulda Gap terrain data base was 12,000 meters. So in this case the SCALE factor was 240. The x coordinate value used by NPSNET was the difference

between the xcoordinate read and the minimum x value, multiplied by the SCALE factor. This x coordinate value was stored in a buffer called npsnetxunit. The contents of npsnetxunit were the next field written to npsscriptin.

(5) Vertical coordinate data was given the value of zero at this point since NPSNET determined the vertical coordinate based on system type and location on the terrain. This zero value was then written to npsscriptin.

(6) The y coordinate transformation was the same as was done for the x coordinate. And in the case of scenario numbered 716, the SCALE factor was exactly the same: 240. The y coordinate value was stored in the buffer npsnetyunit and written to the script file.

(7) The orientation data was read into a buffer called dview. The content of dview was then written to the script file.

(8) The same orientation angle was written again as the value for the weapon orientation of the system.

(9) The next field in the script file was elevation above ground. The value written here was based on which side the system belonged to, and whether the system was a ground vehicle, a helicopter, or an airplane.

(10) The gun elevation data buffer gun_elev was given the value zero and written to the script file.

(11) The number of rounds fired did not change in this application so the value of zero was the default written on each script line for this field.

(12) All the vehicles were given the default value of one in the last field of each script line which indicated to NPSNET that the system was alive.

b. Processing subsequent node data

Subsequent node information was processed by the inner loop of writescriptin. Each system had a possibility of fifty subsequent nodes. Unused nodes had -1.0 entered for the values of their x and y coordinates. Each iteration of the inner loop tested for x and y values greater than -1.0. If either value was -1.0 then a for loop was entered which iterated the number of times equal to the number of path node spaces not yet used. Each iteration of the loop advanced the pointers in the three files which contained subsequent node data: xnode_dat, ynode_dat, and tnode_dat. After the conclusion of the loop, a break statement moved control to the outer loop and the next script line was scanned to determine if it contained useful initial node information.

Except for the hours, minutes, seconds, x coordinate, and y coordinate, all the other values for the system stayed the same. So only those five excepted fields were given updated values. All the other fields received the same values that were given to the initial node script line. The order and number of fields stayed the same whether or not the script line was for an initial or subsequent node.

3. Calculating direction and speed

After all the script lines were written to the script file npsscriptin, the file was closed. Then a call to the function calculate_direction_and_speed was made. The final processes called by writescriptin are shown in Figure 7. The values entered for the direction and speed of the systems were only valid for the initial node. Speeds and orientations of subsequent nodes of a system path had to be calculated next and written into the proper fields of the script lines.

This was accomplished by determining which script lines represented nodes on a path. To do this, two lines of the script file npsscriptin were read and compared. If the first line had a different system number than the subsequent line, then they were not related. And if the first script line was the last node of a path, then the values for orientation and speed at the last node of the path were given the same values as those given at the next to the last

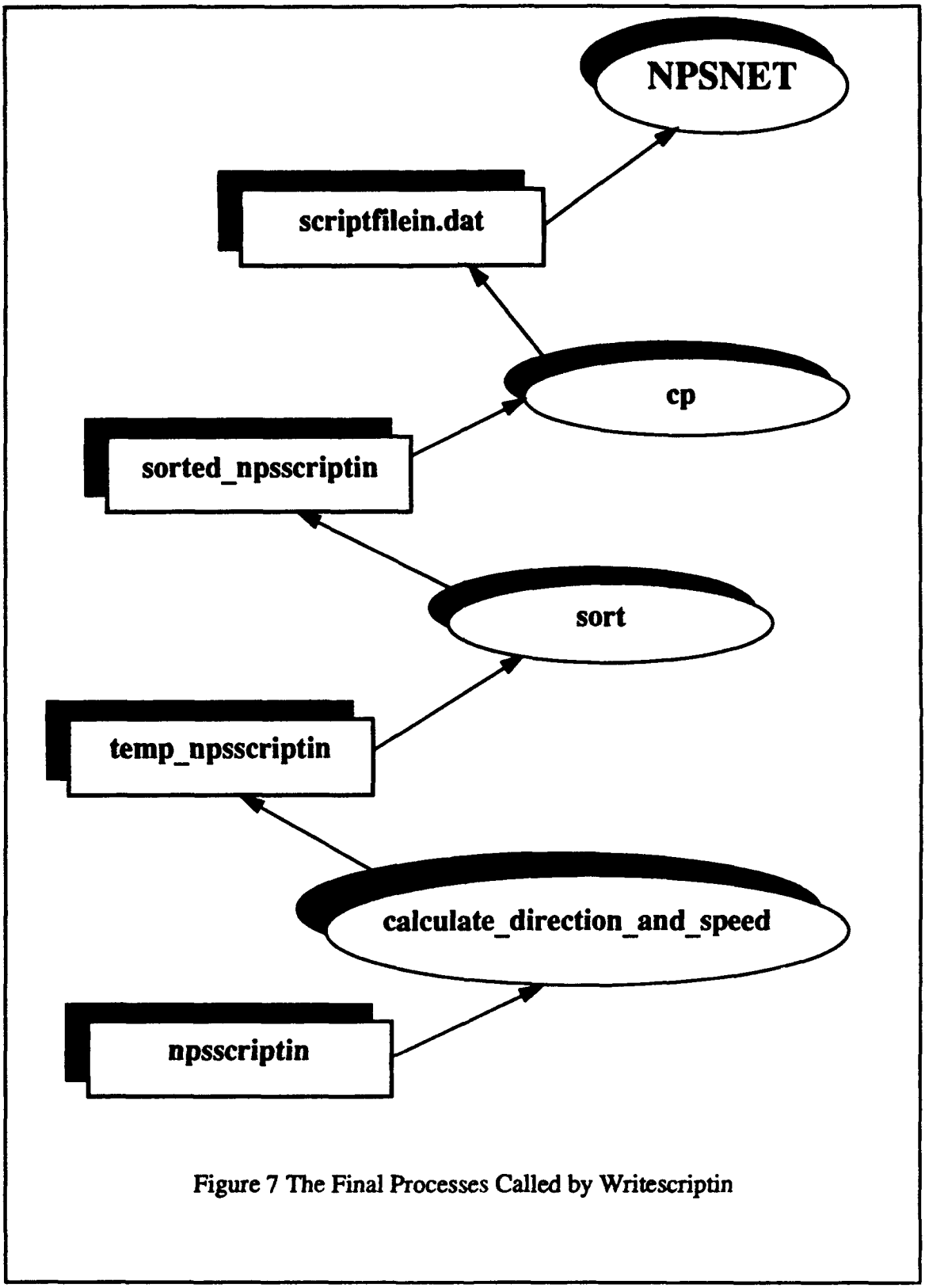

Figure 7 The Final Processes Called by Writescriptin

node of the path. If the two script lines were not related and the first script line was not the last node of a path, then only the orientation had to be converted to degrees - the speed remained the one half maximum speed already given in writescriptin.

If the two script lines were of the same system path then the speed and orientation between the two nodes had to be calculated. The radian orientation was determined using the system arc tangent function fatan2 which took two floating point arguments: the difference between the two nodes along the z-axis, and the difference between the two nodes along the x-axis. The orientation in radians was then converted to degrees. The speed, in meters per second, was determined by dividing the distance between the two nodes by the difference in time between the nodes.

In order to compare each script line with its next script line, the file pointer position at the beginning of the subsequent script line read had to be saved. That way the pointer could be reset after the subsequent script line read. This was necessary since during the next iteration of the loop the subsequent script line would be the first script line read. The two system functions used for this file pointer manipulation were ftell and fseek. Ftell returned an integer offset from the beginning of the file. This offset was the argument to the fseek call.

4. Sort

This system command sorted the npsscriptin file in time order. By giving the following command:

system("sort -o sorted_npsscriptin +1 -3 npsscriptin").

The unix sort program sorted the npsscriptin script lines according to the first three fields of each script line, and wrote the result to the script file called sorted_npsscriptin. At this stage, all the data needed to display the Janus scenario on NPSNET was contained in the script file.

5. Copy

Sorted_npsscriptin was copied to the script file scriptfilein.dat which was read by NPSNET. The following system command at the end of writescriptin copied the script file.

system("cp sorted_npsscriptin /n/gravy1/work2/pratt/simnet/sdis/coll2/vehposfiles/scriptfilein.dat").

After the above processes had been run, NPSNET was started, a vehicle was selected and moved about the virtual battlefield which caused script lines to be written to a file called scriptfileout.dat.

V. TRANSLATING AN NPSNET SCRIPT INTO JANUS BINARY DATA

NPSNET generated a script line each time a system changed its status in speed, or orientation. These script lines were written to the script file called scriptfileout.dat. The next step in the Janus/NPSNET cycle was converting the scripted data back into Janus' binary format and then writing the data to the proper places in the Janus scenario arrays. The name of the main process in the conversion of data from NPSNET to Janus was writedatafiles shown in Figure 8.

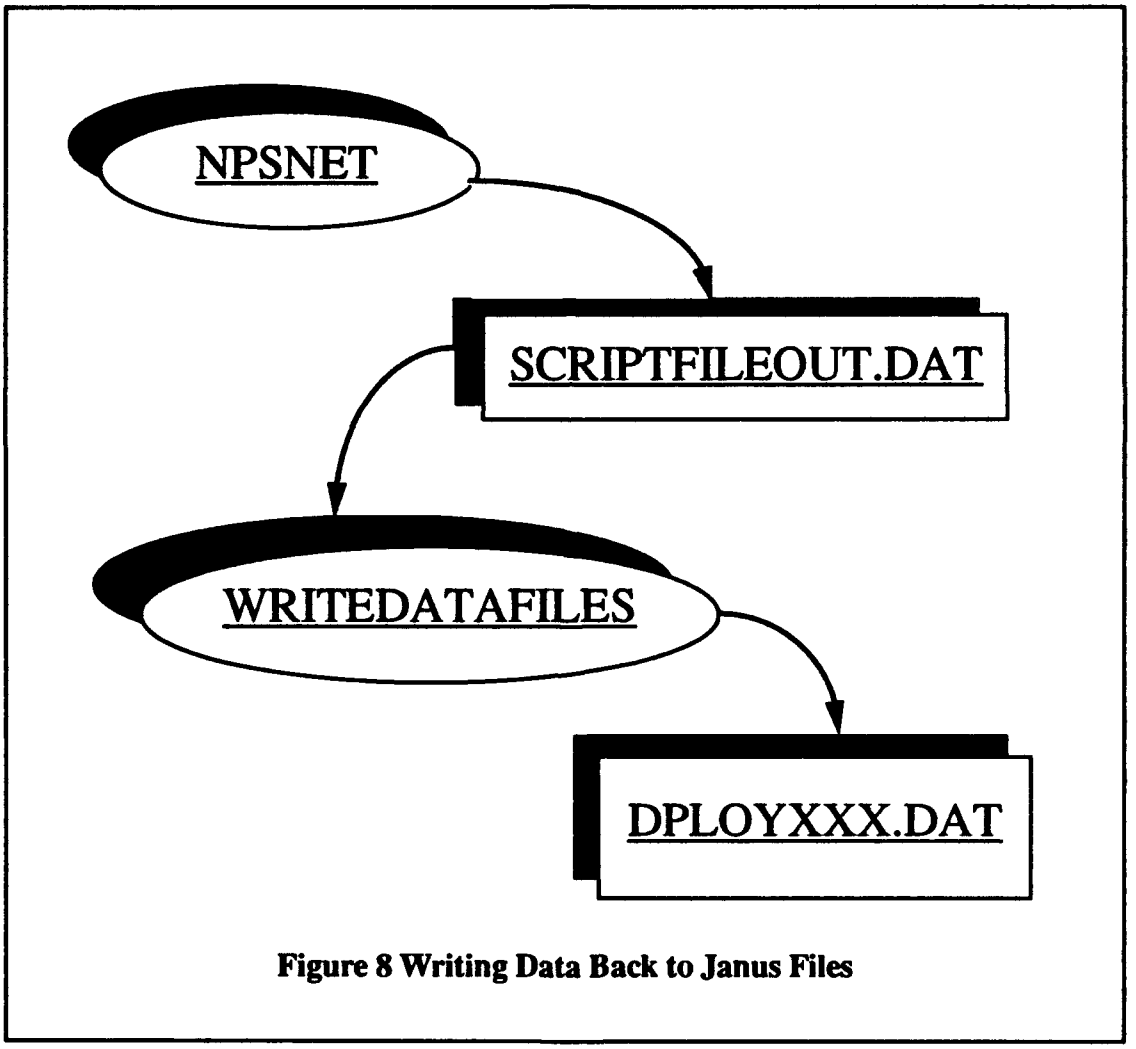

Figure 8 Writing Data Back to Janus Files

A. WRITEDATAFILES

Scriptfileout.dat was a history in chronological order of the movements of systems. Each time the status of a vehicle changed in orientation or speed, a script line was generated. Not all the script lines generated by NPSNET were necessary to describe new system paths in the Janus files. Some of the data had to be ignored, but the scripted data was not ordered in a way that made it easy to distinguish useful script lines. The first processes in writedatafiles changed the order of the script lines and the order and format of the data fields in each script line.

The first process in writedatafiles was a system call to the process sort_by_vehno. The script lines in scriptfileout.dat were sorted chronologically, but subsequent processes required that the script lines be sorted in vehicle number order and then in chronological order. The process sort_by_vehno moved the vehicle number field from the third place to the first place on the script line and then sorted the scriptlines using the system sort program. The system sort program was used for the convenience of not writing a special sort routine for this application. The system sort program took the named file argument and sorted in the order of the key fields designated in the system call. It was possible to specify that the sorting be done on several key fields by designating a starting field and an ending field before which sorting would stop.

Setting up the file for sorting was accomplished by moving the vehicle number field to the first place on the script line and designating it the starting field and then putting the hours, minutes, and seconds fields next; and designating the field following the seconds field as the ending field. The result was that all the script lines with the same system number were group together chronologically. The format of the script lines in scriptfileout.dat, and the format of the script lines in npsscriptout after the call to sort_by_vehno are shown in Figure 9.

A problem with using the system sort program was later discovered. Even though an ending field was designated, if all the values in the key fields were equal to the previous line values, then subsequent fields were used to sort the script lines. This happened when a

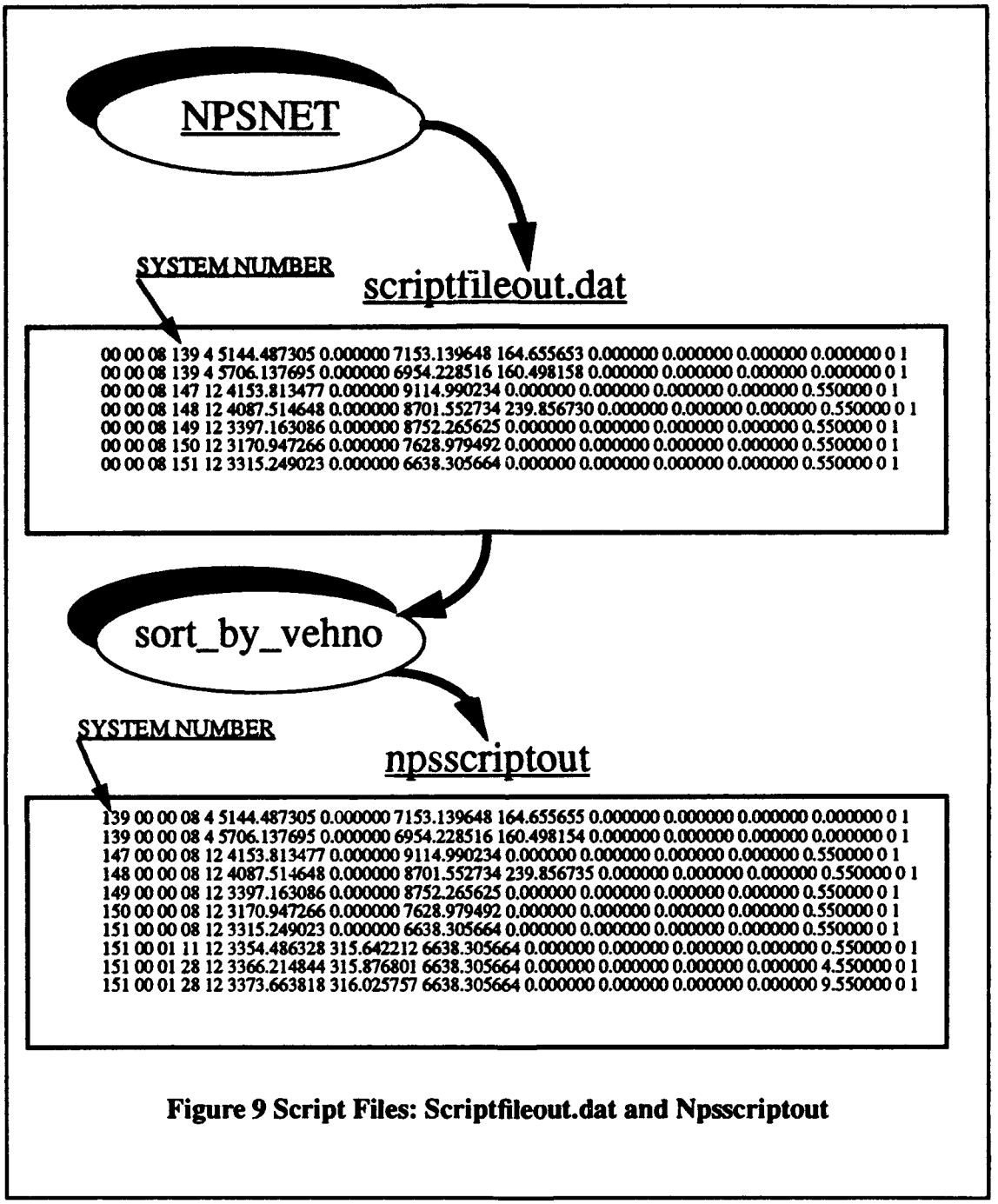

Figure 9 Script Files: Scriptfileout.dat and Npsscriptout

system in NPSNET was made to turn more than 10 degrees in a second. When that was done, script lines were generated that had the same values in the first four fields and the tie was broken by the x coordinate field. This was not a desirable effect because even though two script lines had the same values in their first four fields the order in which they were

written preserved the continuity of the points along the path of the system. Ordering of the x coordinate points alone did not preserve that continuity. This problem with the system sort routine made it necessary to insure that a system did not turn more than ten degrees per second.

After the completion of the system call to sort_by_vehno, the system returned to writedatafiles where a fscanf of maxandmin_dat was called to get the maximum and minimum Janus x and y coordinates previously determined. These were used to convert NPSNET coordinates back into Janus coordinates.

B. EXTRACTING DATA FROM THE SCRIPT LINES

The main purpose of writedatafiles was to determine the path nodes of the system that was maneuvered in NPSNET, and to write the node data into the proper locations in the arrays in the Janus scenario files. All the script lines in npsscriptout were read by the process writedatafiles. Each line was important in determining the path of the system maneuvered in NPSNET. However, not all the script lines contributed data, and not all the data in contributing script lines was written to the Janus binary files.

The path of a system in the Janus scenario was, for the purposes of this thesis, an initial position and a series of nodes (recall that speed and orientation calculations were done in writescriptin). Each node had an x and y coordinate, and a time in seconds. Each NPSNET script line referred to one vehicle and the script line had x and z coordinates, and a time in hours, minutes and seconds. The x and y coordinates and the total time since the beginning of the scenario were the three data values that had to be written to the proper places in the Janus binary file DPLOYXXX.DAT. The first line of npsscriptout was read, then a loop was entered which read each npsscriptout script line until the end of file.

At the beginning of each loop iteration, the total time in seconds was calculated from the hours, minutes, and seconds buffers. Then the difference in time since the last script line was read was calculated. Next the system numbers of the current and last iteration of the

loop were compared. If the system numbers were the same, then the current script line represented a part of a turn on the system path.

If the system numbers of the two script lines compared were not the same, then the current script line was the initial node of a new system. And if the previous script line was a subsequent script line of a system path, then the previous script line's data, which was stored in an array, was written to the Janus binary file, DPLOYXXX.DAT. Then the current script line's data was written to DPLOYXXX.DAT. If the previous script line was not part of a path, then just the current script line was written as the initial node of a system.

A system that was maneuvered in NPSNET usually generated many script lines: each time the speed or orientation of the vehicle changed a script line was generated. Going back to the case when script lines were parts of a system path in which turns were represented, not all of these script lines could be written to DPLOYXXX.DAT or the maximum of fifty nodes allowed in the Janus arrays would have been quickly reached. So a single turning point was selected that would represent all the many incremental turns written in the script lines. The turning point information written to DPLOYXXX.DAT was the x and y coordinates of the turn, and the time when the turn took place.

The problem of finding a representative turning point from among the incremental turns along the turn path was solved by not selecting one of those points at all. Instead a point outside the turn was determined by resection.The function called find_intersection was taken directly from the movement program used to run an autonomous mobile robot [Kana91]. The function took two arguments, each of which contained coordinate and orientation data for two points. One point was taken from the line leading into the turn and

the other point was taken from the line going out of the turn. From these two lines an intersection point was determined, Figure 10.

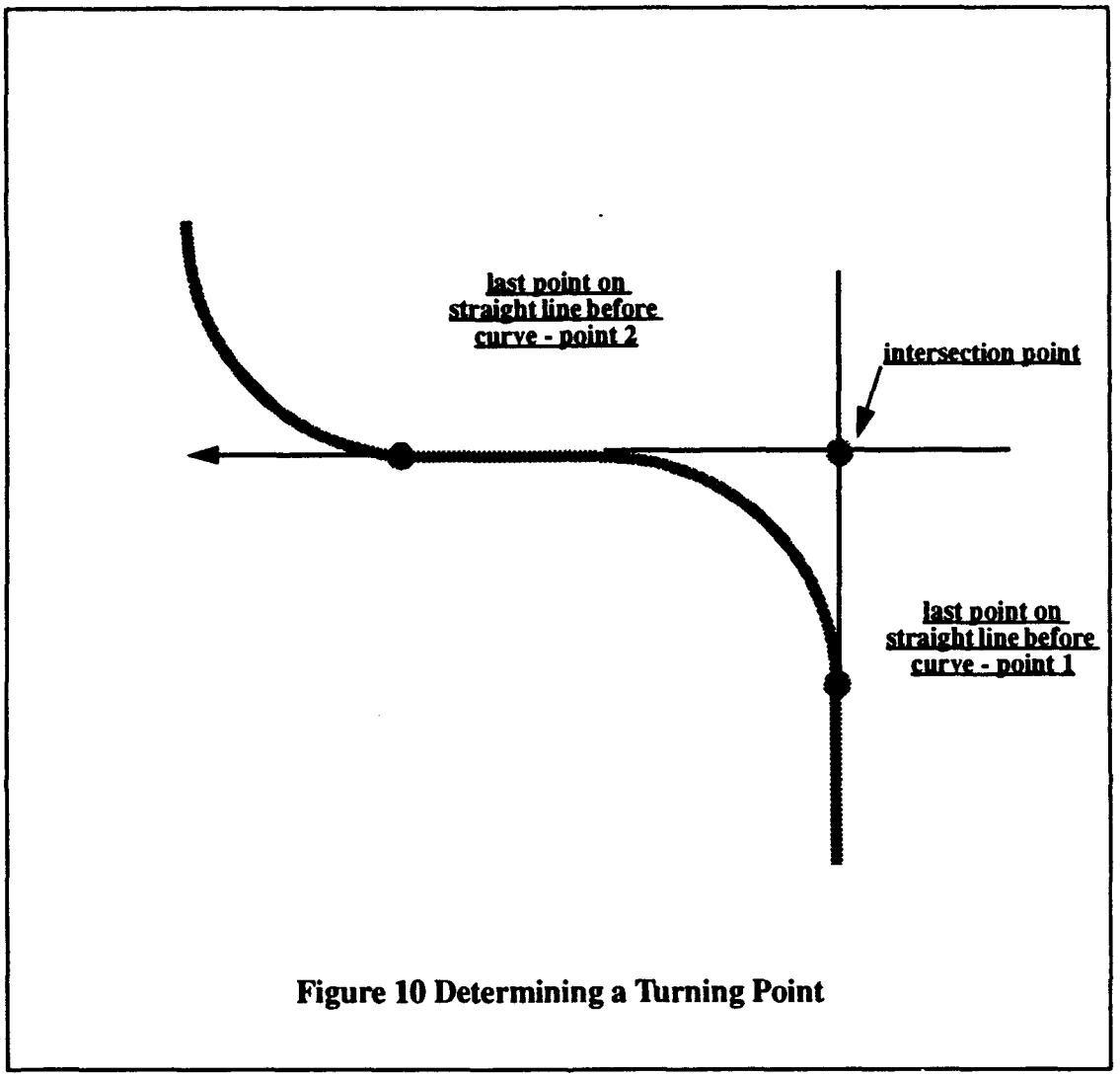

Figure 10 Determining a Turning Point

Selecting the two points required determining when a turn took place. Simply sampling the orientation field of each script line for changes was unsatisfactory. Since if the system was moving between turns in a straight line and not changing speed, then there would be no two script lines with the same orientation to indicate straight line movement. There was no guarantee that other events would generate script lines. Instead the change in time between nodes and a change in system orientation were used to determine when a turn started. If the time between two script lines was greater than five seconds and the system

orientation did not change, then the intersection point was determined. The time used for the node information was the time at which the turn was initiated. Much more accurate and elegant algorithms for selecting a turning point are available from among the algorithms used in robotics research, however the resolution of the Janus system was such that these algorithms served well enough.

C. CONVERTING THE DATA AND WRITING TO JANUS FILES

Before the data was written back into the binary scenario files, the data was converted to the units of measure used in Janus. The x and y coordinates were divided by the SCALE factor and added to their respective minimum values. The time was converted to total seconds at the beginning of each loop.

Writing the algorithm for finding the correct spot to write the data to was the result of calculations and observations of octal dumps of the binary files. The position at which, for example, system number 151 stored the y coordinate of the 27th node of its path was always the same no matter which scenario was used. The scenario files were arrays after arrays of data - there was no dynamic variation in the size of the storage. So the starting points of the arrays written to in numbers of bytes from the start of the file were determined. Knowing the starting point of the array and using the system command lseek, with the whence constant set to SEEK_SET, it was possible to move the pointer to the beginning of an array. Again using lseek, with the whence constant set to SEEK_CUR, the system number was used to calculate the offset in bytes to the point in the array where the data had to be written. Figure11 gives an example of such a calculation.

Given: start of x node array in DPLOYXXX.DAT is the 25236th byte; the system number is 151; and it is the 27th node.

Find the location in the Janus deploy file, DPLOYXXX.DAT, for the data.

Location of the 27th x node of system 151 = **location in binary files of the beginning of the x node array** + (**system number minus one** * **the fifty nodes alloted each system** + **the node count**) * **four bytes**

Location of the 27th x node of system 151 = 25236 + (150 * 50 + 27) * 4 = 55344

Figure 11 Example: Determining a Location in Binary Files

VI. CONCLUSIONS AND FUTURE WORK

It was possible to take a two-dimensional Janus scenario written in a binary format, convert it to script files in ASCII format and play it in the three-dimensional virtual battlefield of NPSNET. Thus providing a user the capability of viewing the battlefield in three-dimensions and maneuvering a system in a more realistic environment. Further, the resulting movements upon systems on the virtual battlefield provided by NPSNET could be stored and written back into the Janus scenario. Thus a scenario could be completely constructed, or existing scenarios could be modified in a more realistic NPSNET environment.

Future work on this application of NPSNET could include more realistic turning of vehicles. The integration of an algorithm to generate smooth turns with changes in velocity during and after the turn would be more realistic than the current version which simply moves to a point and abruptly changes direction. Also the system speed should be modified just before entering the turn and after leaving the turn. Many such algorithms are available in the movement control routines of the autonomous mobile robot Yamabico. The Yamabico code is written in "C" and has been tested and it should be easy to assimilate the Yamabico code into NPSNET. [Kana91]

These algorithms could be integrated into the calculate_direction_and_speed process which was called at the end of the writescriptin process. Janus was limited to only fifty nodes for a system path, but NPSNET is not so restrictive. Using the Yamabico code, many script lines could be generated to describe a smooth turn with changes in speed which could be written to the input script file.

Yet another improvement of this application could take a single vehicle path and generate script lines for a formation of vehicles following along the same path. It seems that the complexity of this problem could be handled by considering the formation as a wire-frame model, where each vertex of the wire frame represents a vehicle in the formation. Then projective transformations (forward kinematics in robotics jargon) could be used to

calculate the locations of the vertices of the wire-frame model in Cartesian coordinates. The rotations at the vertices could be done by Euler angle transformations. A "C++" version of these methods was written by Sandra Davidson in an Naval Postgraduate School thesis for a graphic simulation of a walking robot [Davi 93].

A list of unused system numbers would have to be made at the beginning of the process writescriptin. Then the available system numbers could be assigned to the vertices of the polygon representing the formation of systems. The transformations would only have to be done in two dimensions since NPSNET uses the terrain elevations to provide the elevation data of the systems.

While the use of a three-dimensional virtual battlefield was clearly more realistic, and so worth pursuing, another area of research that could be explored with this application is the question of how much a three-dimensional view of the a virtual battlefield would help in the selection of courses of action. It seemed self evident that a three-dimensional view would be an enhancement, but how much better would it be?

APPENDIX A: USER GUIDE

A. INTRODUCTION

This chapter provides a step-by-step description of how to add vehicle paths to a Janus binary database using NPSNET. Figure 12 lists the files required to run this application.

Executables:

goldtest/deploy/readsundeploy

goldtest/force/readsunforce **calculate_direction_and_speed**

goldtest/system/readsunsystem **sort_by_vehno**

writescriptin **writedatafiles**

goldtest/deploy/janustonpsscript_dat/maxandmin

Janus Data Files:

DPLOYXXX.DAT **FORCXXX.DAT** **SYSTMXXX.DAT**

Data Files in goldtest/deploy/janustonpsnet_dat/

xunit_dat	**xnode_dat**	**tnode_dat**
yunit_dat	**ynode_dat**	**dview_dat**
system_dat	**maxandmin_dat**	

Data Files in goldtest/deploy/:

npsscriptin **npscriptout**

sorted_npsscriptin **sorted_npsscriptout**

temp_npsscriptin

Directories with Makefiles:

~smithrs/thesis/goldtest

~smithrs/thesis/goldtest/deploy

~smithrs/thesis/goldtest/force

~smithrs/thesis/goldtest/system

~smithrs/thesis/goldtest/deploy/janustonpsscript_dat

Figure 12 Files Required for 3-D Script Generation

Figure 13 shows an example of how to run the programs developed in this thesis.

```
> cp DPLOY716.DAT ~smithrs/thesis/goldtest/deploy
> cp FORC716.DAT ~smithrs/thesis/goldtest/force
> cp SYSTM716.DAT ~smithrs/thesis/goldtest/system
> cd ~smithrs
> cd deploy
> readsundeploy DPLOY716.DAT
> cd ../force
> readsunforce FORC716.DAT
> cd ../system
> readsunsystem SYSTM716.DAT
> cd deploy/janustonpsnet_dat
> maxandmin
> cd ../..
> writescriptin
> npsnet
(From the popup menus select the following.)
    - 2D Map Window
     - Display 2d Map
    - Script Menu
     - Record and Play
(Entities on the screen are manuevered at this point and the movements are
stored to a script file.)
    - Exit
> writedatafiles
(Scripted movements are now stored in the original Janus scenario. To view
the new movements on NPSNET proceed with the following commands.)
> cd deploy
> readsundeploy DPLOY716.DAT
> cd ..
> writescriptin
> npsnet
(From the popup menu select the following.)
    - 2D Map Window
     - Display 2d Map
    - Script Menu
     - Play
```

Figure 13 Example Program Run of Janus Scenario 716

B. BINARY TO ASCII FILES

First, a Janus scenario of four files is required: they will be DPLOYXXX.DAT, FORCXXX.DAT, SYSTMXXX.DAT and JSCRNXXX.DAT, where the XXX refers to a

three-digit number associated with a particular scenario. The executable files which read the scenario files are in the same directories and are called readsundeploy, readsunforce, and readsunsystem: recall that JSCRNXXX.DAT is not used. The three previous programs produce seven ASCII files which store the data in files called xunit_dat, yunit_dat, xnode_dat, ynode_dat, tnode_dat, dview_dat, system_dat. These files are found in /n/gravy1/work3/smithrs/thesis/goldtest/deploy/janustonpsnet_dat/.

C. ASSEMBLING THE INPUT SCRIPT FILE

The maximum and minimum values of all the x and y coordinates in the scenario must also be determined. Maxandmin produces the file maxandmin_dat in which the maximum and minimum x and y values are stored. The process file maxandmin and the data file maxandmin_dat are found in /n/gravy1/work3/smithrs/thesis/goldtest/deploy/janustonpsnet_dat. These minimum values are used by other processes to determine the scale of the scenario when displayed on NPSNET. Since it is unlikely that paths added to the scenario will change the minimum values, it is necessary to run this process only once. The script file which is the input to NPSNET is produced by calling the process file writescriptin found in /n/gravy1/work3/smithrs/thesis/goldtest. The script file which NPSNET will read is called scriptfilein.dat and is found in /n/gravy1/work2/pratt/simnet/sdis/coll2/vehposfiles.

D. RUNNING NPSNET

To run NPSNET go to the directory, /n/gravy1/work2/pratt/simnet/sdis/coll2. Type npsnet w. The argument w allows the user to choose the size of the window. Once the terrain is displayed, hold down a right mouse button click and highlight the "2D Map Window" from the popup menu. Another popup menu will appear from which "Display 2d Map" should be highlighted. This causes a two-dimensional map to be displayed in the upper right-hand corner of the screen.

Next, the input script file must be started and displayed. Again holding down the right mouse button in an NPSNET window, highlight "Script Menu". Another popup menu will

appear and "Record and Play" must be selected. This will open both the input and output script files used by NPSNET and start the scripted Janus scenario stored in the input script file.

System selections are made by moving the cursor into the two-dimensional screen area, selecting a system icon, and pressing the left mouse button. A few seconds will pass before the view presented on the three-dimensional screen is that of the system selected. Now the user can move the system around the battlefield interacting with the terrain and other systems from the three-dimensional point of view of the system selected. Movement is controlled by arrow keys, spaceball, or dials depending on what is available at the terminal.

Another feature added to the NPSNET/Janus interface was the ability to select a system icon from the two-dimensional screen with the mouse and, while holding down the mouse button, to move the icon to another location. This movement generated another script line in the output script file, and so it was recorded to the Janus scenario as a starting location if no other script lines came before it.

E. STORING THE NEW PATH DATA TO THE JANUS DATABASE

A fter the scripting option was selected, all the while that NPSNET was running it was producing a script file called scriptfileout.dat. The file is in /n/gravy1/work2/pratt/simnet/sdis/coll2/vehposfiles. The process file which reads the data, converts it, and writes it to the Janus database is called writedatafiles. It is found in the directory /n/gravy1/work3/smithrs/thesis/goldtest. After running this process the new path data is stored in the Janus deploy file, DPLOYXX.DAT.

F. DISPLAYING THE NEW PATH DATA

The new path data can be displayed with some of the same process files that created the initial scripted version of the Janus scenario: the processes readsunsystem, readsunforce, and maxandmin does not need to be called again.

So the first process to call is readsundeploy which is found in /n/gravy1/work3/smithrs/thesis/goldtest/deploy, and then writescriptin found in /n/gravy1/work3/smithrs/thesis/goldtest - recall that maxandmin doesn't need to be run again. Now the NPSNET input script file is ready for display. Start NPSNET and bring up the two-dimensional map in the same way as before. Next select the "Script Menu" from the popup menu in the same way as before except that instead of selecting "Record and Play" from the second popup menu, select "Play Tracks". After the scenario is displayed the user can watch the newly scripted system from another system or select the system in the same way as before and watch the scenario unfold from the point of view of the newly scripted vehicle.

G. TRANSPORTING THE FILES TO OTHER DIRECTORIES

Take the Makefiles along with their respective process files. The storage file names, with their entire path, are usually referred to at the top of the process files which write or read them. So if the files are moved it is necessary to change the #define calls which specify the paths.

LIST OF REFERENCES

[Davi93] Davidson, Sandra Lynne, "An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Walking Robot Kinematics", Naval Postgraduate School, Monterey, CA, September,1992.

[Kana91] Kanayama, Yutaka, "Advanced Robotics Course Notes", Naval Postgraduate School, Monterey, CA, Fall 1991.

[SGI91] Silicon Graphics, Inc., "Graphics Library Programming Guide", Document Number 007-1210-040, Mountain View, CA, 1991.

[Walt92] Walter, Jon C., and Warren, Patrick T., "NPSNET: Master's Thesis in Computer Science, JANUS-3D Providing Three-Dimensional Displays for a Traditional Combat Model", Naval Postgraduate School, Monterey, CA, September,1992.

[Zyda92] Zyda, Michael J., Pratt, David R., Monahan, James D., and Wilson, Kalin P., "NPSNET: Constructing a 3D Virtual World.", *1992 Symposium on Interactive 3D Graphics*, 30 March, 1992. pp. 147-156.

[Zyda93] Zyda, Michael J., Wilson, Kalin P., Pratt, David R., Monahan, James G. and Falby, John S. "NPSOFF: An Object Description Language for Supporting Virtual World Construction", *Computers & Graphics*, accepted for Vol. 17, No. 4, 1993.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943-5002	2
Dr. Michael J. Zyda Computer Science Department Code CS/ZK Naval Postgraduate School Monterey, CA 93943	2
Dr. David R. Pratt Computer Science Department Code CS/PR Naval Postgraduate School Monterey, CA 93943	2
Director TRAC Monterey PO Box 8692 Monterey, CA 93943-0692	1
CPT Richard S. Smith US Army Intelligence Center and School Fort Huachuca, AZ 85613	1
Dr. Ted Lewis Professor and Chairman Computer Science Department Code CS/Lt Naval Postgraduate School Monterey, CA 93943	1